PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS

Engenharia de Computação
Trabalho Interdisciplinar de Internet das Coisas

SISTEMA INTELIGENTE DE ALIMENTAÇÃO E HIDRATAÇÃO PARA PETS

Bárbara Maria Sampaio Portes
Samir da Morim Cambraia

Belo Horizonte
30 de junho de 2025

Sumário

1 Introdução

Este documento detalha o desenvolvimento do Trabalho Interdisciplinar (TI) da disciplina de Internet das Coisas, que consistiu na criação de um sistema completo para o monitoramento e automação do fornecimento de alimento e água para animais de estimação. A solução foi projetada seguindo a arquitetura padrão de uma aplicação IoT, abrangendo desde os dispositivos de hardware (sensores e atuadores) e sua programação, passando pela comunicação em nuvem com o protocolo MQTT, até a persistência de dados em banco de dados e a visualização e controle por meio de um aplicativo mobile.

O projeto visa solucionar um problema cotidiano de muitos tutores de pets: garantir a alimentação e hidratação adequadas de seus animais, mesmo quando estão ausentes. Através da tecnologia, buscamos oferecer uma solução que trouxesse segurança e tranquilidade, permitindo o acompanhamento e a interação remota com o sistema.

Ao longo do semestre, cada etapa da arquitetura foi implementada e validada através dos trabalhos parciais, culminando em um sistema funcional que, embora não tenha tido sua montagem de hardware finalizada, teve todos os seus componentes de software e simulação desenvolvidos e testados com sucesso, provando a viabilidade do conceito.

2 Objetivos

2.1 Objetivo Geral

Desenvolver um sistema IoT funcional e integrado que automatize o monitoramento e o fornecimento de alimentação e hidratação para pets, garantindo seu bem-estar e oferecendo controle remoto e tranquilidade aos tutores.

2.2 Objetivos Específicos

- **Monitorar** continuamente o nível de ração e água disponíveis nos recipientes do pet.
- **Automatizar** a reposição de água e a liberação controlada de ração, seja por programação ou sob demanda.
- **Transmitir** em tempo real os dados coletados pelos sensores para uma plataforma em nuvem, utilizando o protocolo MQTT.
- **Alertar** o tutor através de notificações em caso de condições críticas, como reservatórios vazios ou níveis de temperatura inadequados.
- **Desenvolver** uma interface de controle em um aplicativo mobile para visualização dos dados, envio de comandos manuais e configuração de alertas.
- **Armazenar** o histórico de dados de sensores e atuadores em um banco de dados para futuras consultas e análises.

3 Motivação

A motivação para este projeto nasceu da observação de uma necessidade real e crescente entre os tutores de animais de estimação. A rotina agitada e os períodos de ausência, seja por trabalho ou viagens, geram preocupação com o cuidado contínuo dos pets. A falta de um sistema para

garantir que eles tenham acesso constante a alimento e água fresca pode levar a problemas de saúde, como desidratação ou superalimentação.

A integração de dispositivos de Internet das Coisas (IoT) oferece uma solução elegante e eficaz para este problema. A possibilidade de monitorar e atuar remotamente não só resolve a questão prática, mas também fortalece o vínculo entre tutor e pet, oferecendo paz de espírito. Este projeto foi, portanto, motivado pelo desejo de aplicar a tecnologia de forma significativa para melhorar a qualidade de vida tanto dos animais quanto de seus donos.

4 Projeto de Hardware

A montagem física final do hardware não foi concluída. No entanto, todo o sistema foi planejado e sua funcionalidade foi validada por meio de um protótipo em protoboard, que utilizou LEDs e botões para simular o comportamento dos sensores e atuadores reais. Esta abordagem permitiu o desenvolvimento e teste completo da lógica de software e da comunicação com a nuvem.

4.1 Diagrama Elétrico

O diagrama abaixo ilustra a arquitetura de hardware que foi planejada para a versão final e completa do projeto. Ele detalha a conexão entre o microcontrolador ESP32, os sensores (temperatura, ultrassônico) e os atuadores (drivers de motor, motor de passo).

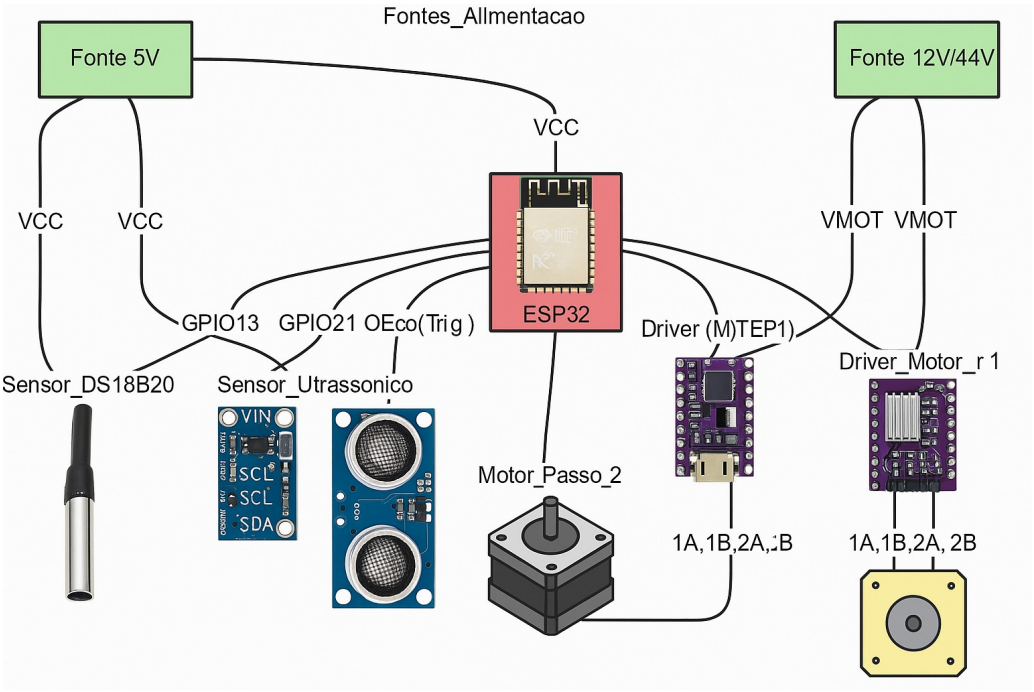

Figura 1: Diagrama elétrico do projeto Pet Feeder.

4.2 Protótipo de Validação

A imagem a seguir mostra o circuito montado em protoboard utilizado para os testes e desenvolvimento dos Trabalhos 2 e 5. Nele, botões com resistores de pull-up simularam os sinais dos sensores, e LEDs indicaram a ativação dos atuadores, validando a lógica do firmware e a comunicação MQTT.

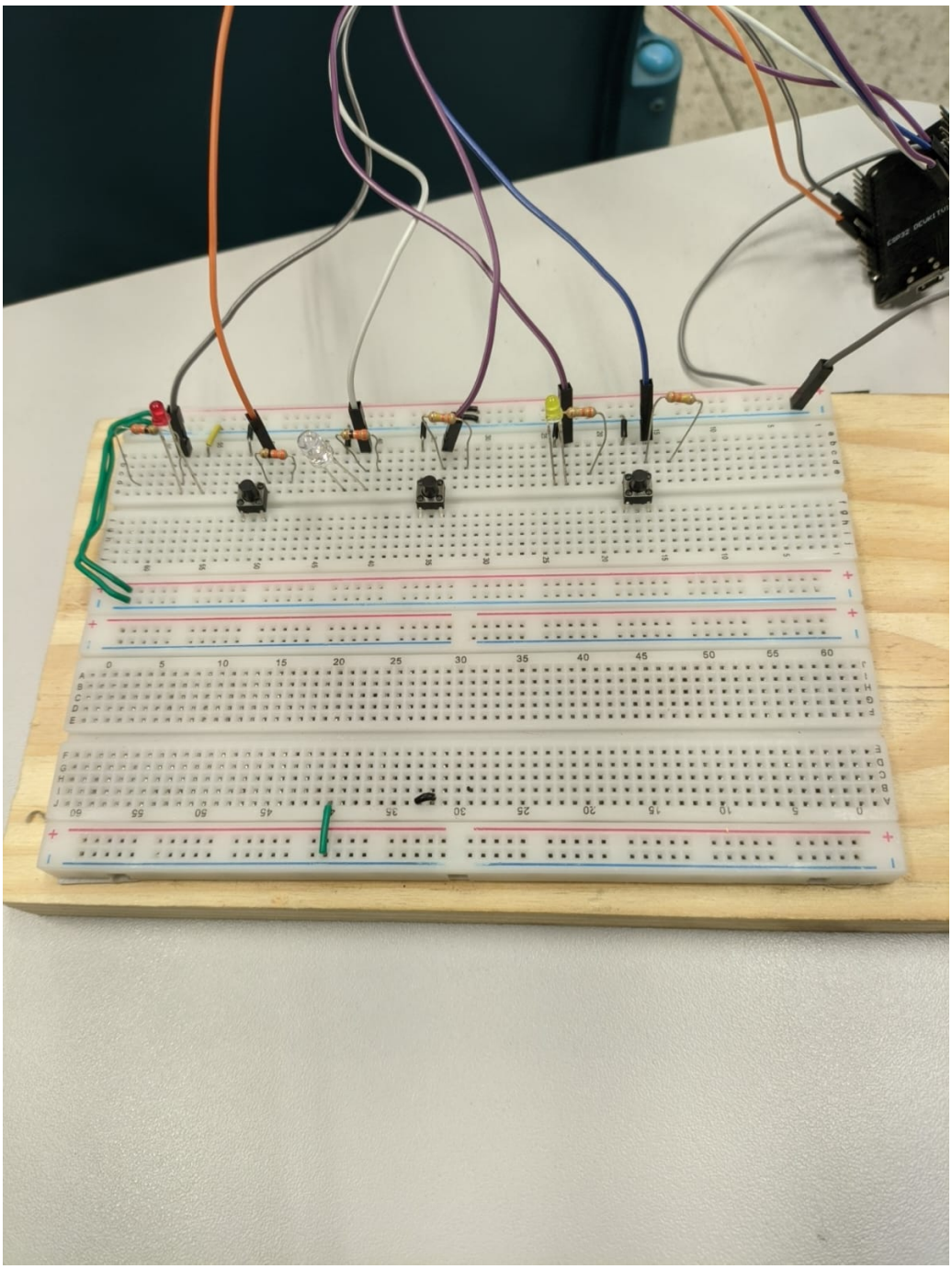

Figura 2: Protótipo em protoboard para simulação de sensores e atuadores.

4.3 Descrição dos Componentes

Sensor Ultrassônico (HC-SR04): Este sensor seria utilizado para medir o nível de ração no reservatório. Ele funciona emitindo um pulso sonoro de alta frequência e medindo o tempo que o eco leva para retornar. Com base nesse tempo, o ESP32 calcularia a distância até a superfície da ração, determinando se o nível está baixo e se é necessário alertar o tutor.

Sensor de Temperatura (DS18B20): Este sensor à prova d'água seria posicionado no recipiente de água para monitorar sua temperatura. Caso a temperatura subisse a um nível inadequado para o consumo do pet, o sistema poderia acionar a bomba d'água para renová-la e enviar um alerta para o aplicativo do tutor.

Motor de Passo (28BYJ-48) e Driver: O motor de passo seria o atuador responsável por liberar a ração. Ao receber um comando do aplicativo ou de uma programação, o ESP32 acionaria o motor para girar por um número específico de passos, liberando uma porção controlada de ração no comedouro.

Bomba d'Água (Atuador): Uma pequena bomba de água seria acionada para renovar a água do bebedouro. Isso ocorreria quando o sensor de temperatura indicasse água quente ou quando o tutor enviasse um comando manual pelo aplicativo.

5 Projeto de Software

O software do projeto é dividido em três componentes principais: o firmware do ESP32, o script de backend para persistência de dados e o aplicativo mobile para interação com o usuário.

5.1 Comunicação via Protocolo MQTT

A comunicação entre todos os componentes de software do sistema é realizada através do protocolo MQTT (Message Queuing Telemetry Transport), um padrão leve e eficiente para troca de mensagens, ideal para dispositivos IoT. A arquitetura se baseia nos seguintes componentes:

Broker: É o servidor central que gerencia o fluxo de mensagens. Neste projeto, utilizamos o broker público `test.mosquitto.org`. Todos os clientes se conectam a ele para enviar ou receber informações.

Clients (Clientes): São os dispositivos e aplicações que se conectam ao broker. Em nosso sistema, temos três clientes principais:

- O microcontrolador **ESP32**, que publica os status dos sensores e se inscreve para receber comandos.
- O **script Python**, que se inscreve em todos os tópicos para registrar os dados no banco de dados.
- O aplicativo **IoT MQTT Panel**, que se inscreve nos tópicos de alerta para exibir o status e publica nos tópicos de comando para controlar o dispositivo.

Topics (Tópicos): São os "canais"para onde as mensagens são enviadas. Utilizamos uma estrutura hierárquica para organizar os dados, como `/pet/alerta/racao` ou `/pet/comando/agua`. Essa estrutura permite uma gestão clara e escalável das informações.

Publish (Publicar): A ação de enviar uma mensagem para um tópico. Por exemplo, o ESP32 publica a mensagem "FALTA"no tópico `/pet/alerta/racao`.

Subscribe (Inscrever-se): A ação de um cliente informar ao broker que deseja receber as mensagens de um determinado tópico. Por exemplo, o aplicativo se inscreve no tópico `/pet/alerta/racao` para saber quando a ração acaba.

5.2 Fluxograma e Descrição das Funcionalidades

O fluxograma a seguir, representado de forma descritiva, apresenta a operação geral do sistema, desde a coleta de dados até a interação com o usuário.

Firmware (ESP32): O código embarcado no ESP32 é responsável por conectar o dispositivo à rede Wi-Fi e ao broker MQTT. Ele simula a leitura de sensores através de botões. Ao detectar um "nível baixo"(botão pressionado), publica uma mensagem de alerta em um tópico MQTT específico. Ele também fica à escuta de comandos vindos do aplicativo mobile e, ao recebê-los, aciona o atuador correspondente (simulado por um LED).

Backend (Python + Banco de Dados): Um script em Python rodando em um servidor (ou localmente) se conecta ao mesmo broker MQTT e se inscreve em todos os tópicos de alertas e comandos. Ao receber uma mensagem, ele a processa e a insere em um banco de dados MySQL (gerenciado via Docker e hospedado na AWS para escalabilidade). Isso cria um registro histórico de todos os eventos do sistema.

Aplicativo Mobile (IoT MQTT Panel): O usuário interage com o sistema através do aplicativo IoT MQTT Panel. O painel é configurado para exibir o status dos alertas (escutando os tópicos de alerta) e para enviar comandos (publicando nos tópicos de comando), permitindo o controle manual do alimentador.

5.3 Linguagem Utilizada e Justificativa

- **C++ (Arduino Framework):** Utilizado para programar o firmware do ESP32. A escolha se deu pela vasta disponibilidade de bibliotecas (WiFi, PubSubClient), pela facilidade de uso do ambiente Arduino IDE e por ser a linguagem padrão para a prototipação com a plataforma ESP32, conforme abordado na disciplina.
- **Python:** Utilizado para o script de backend que interage com o banco de dados. Python foi escolhido por sua simplicidade, sintaxe clara e pelas excelentes bibliotecas para conexão com MQTT (‘paho-mqtt‘) e bancos de dados (‘mysql-connector-python‘), permitindo um desenvolvimento rápido e eficiente da lógica de persistência de dados.
- **SQL:** Utilizado para a definição da estrutura do banco de dados e para as operações de inserção e consulta. É a linguagem padrão para gerenciamento de bancos de dados relacionais como o MySQL.

6 Custo do Projeto

A tabela abaixo apresenta uma estimativa de custo dos componentes de hardware planejados para a montagem final do projeto, com base em valores de mercado para componentes eletrônicos no Brasil.

Tabela 1: Estimativa de Custo dos Componentes

Componente	**Quantidade**	**Preço Unitário (R$)**	**Preço Total (R$)**
ESP32 Dev Kit	1	R$ 45,00	R$ 45,00
Sensor Ultrassônico HC-SR04	1	R$ 10,00	R$ 10,00
Sensor de Temperatura DS18B20	1	R$ 15,00	R$ 15,00
Motor de Passo 28BYJ-48 com Driver	1	R$ 25,00	R$ 25,00
Driver de Motor A4988	1	R$ 12,00	R$ 12,00
Fonte 5V	1	R$ 20,00	R$ 20,00
Fonte 12V	1	R$ 30,00	R$ 30,00
Protoboard, Jumpers, Resistores	1 (kit)	R$ 35,00	R$ 35,00
Custo Total Estimado			**R$ 192,00**

7 Conclusão

O desenvolvimento do projeto "Sistema Inteligente de Alimentação e Hidratação para Pets"permitiu a aplicação prática e integrada de todos os conceitos fundamentais da arquitetura de Internet das Coisas abordados na disciplina. Desde a programação de baixo nível do microcontrolador até a configuração de serviços em nuvem e a interação com o usuário final, o trabalho proporcionou uma visão completa do ciclo de vida de um produto IoT.

Embora a montagem física do hardware não tenha sido finalizada, o objetivo principal de projetar, desenvolver e validar um sistema funcional foi alcançado com sucesso. A simulação em protoboard e a implementação completa do software demonstraram a viabilidade técnica da solução. A equipe conseguiu integrar o dispositivo com a nuvem via MQTT, persistir os dados em um banco de dados relacional e criar uma interface de controle funcional em um aplicativo mobile.

Este projeto reforçou a importância da metodologia ágil para gerenciar as etapas de desenvolvimento e superou os desafios de integração entre diferentes tecnologias, resultando em um aprendizado significativo e em um protótipo de software robusto e pronto para ser embarcado em um produto final.

A Anexos

A.1 Anexo A: Código Fonte do Firmware (ESP32)

```
// Codigo do Trabalho 5 - Simulacao de Sensores e Atuadores
#include <WiFi.h>
#include <PubSubClient.h>

// Wi-Fi
const char* SSID = "s1";
const char* PASSWORD = "1234567aa";

// MQTT
const char* BROKER_MQTT = "test.mosquitto.org";
const int BROKER_PORT = 1883;
const char* ID_MQTT = "esp32_pet_feeder";

// Topicos MQTT
#define TOPICO_ALERTA_RACAO "/pet/alerta/racao"
#define TOPICO_ALERTA_AGUA "/pet/alerta/agua"
#define TOPICO_ALERTA_TEMPERATURA "/pet/alerta/temperatura"
#define TOPICO_COMANDO_RACAO "/pet/comando/racao"
#define TOPICO_COMANDO_AGUA "/pet/comando/agua"
#define TOPICO_COMANDO_TEMPERATURA "/pet/comando/temperatura"

// Pinos dos botoes (sensores simulados)
#define BOTAO_RACAO 4
#define BOTAO_AGUA 13
#define BOTAO_TEMPERATURA_AGUA 27

// Pinos dos LEDs (atuadores simulados)
#define LED_RACAO 5
#define LED_AGUA 12
#define LED_TEMPERATURA_AGUA 26

// Temporizadores para debounce e controle de envio
unsigned long tempoRacao = 0;
unsigned long tempoAgua = 0;
unsigned long tempoTemperatura = 0;
const unsigned long intervalo = 5000;

WiFiClient espClient;
PubSubClient MQTT(espClient);

void setup() {
    Serial.begin(9600);
    Serial.println("INICIANDO SISTEMA PET FEEDER COM MQTT!");
    pinMode(BOTAO_RACAO, INPUT_PULLUP);
    pinMode(BOTAO_AGUA, INPUT_PULLUP);
    pinMode(BOTAO_TEMPERATURA_AGUA, INPUT_PULLUP);
    pinMode(LED_RACAO, OUTPUT);
    pinMode(LED_AGUA, OUTPUT);
    pinMode(LED_TEMPERATURA_AGUA, OUTPUT);
    conectaWiFi();
    MQTT.setServer(BROKER_MQTT, BROKER_PORT);
    MQTT.setCallback(callback);
}

void loop() {
    unsigned long tempoAtual = millis();
```

```
    if (!MQTT.connected()) {
        reconnectMQTT();
    }
    verificaSensorRacao(tempoAtual);
    verificaSensorAgua(tempoAtual);
    verificaSensorTemperatura(tempoAtual);
    MQTT.loop();
}

void conectaWiFi() {
    WiFi.begin(SSID, PASSWORD);
    Serial.print("Conectando ao Wi-Fi");
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.println("\nWi-Fi conectado!");
    Serial.println(WiFi.localIP());
}

void reconnectMQTT() {
    while (!MQTT.connected()) {
        Serial.print("Conectando ao MQTT...");
        if (MQTT.connect(ID_MQTT)) {
            Serial.println("Conectado!");
            MQTT.subscribe(TOPICO_COMANDO_RACAO);
            MQTT.subscribe(TOPICO_COMANDO_AGUA);
            MQTT.subscribe(TOPICO_COMANDO_TEMPERATURA);
        } else {
            Serial.print("Erro: ");
            Serial.println(MQTT.state());
            delay(2000);
        }
    }
}

void callback(char* topic, byte* payload, unsigned int length) {
    String msg;
    for (int i = 0; i < length; i++) {
        msg += (char)payload[i];
    }
    Serial.print("Comando recebido [");
    Serial.print(topic);
    Serial.print("]: ");
    Serial.println(msg);
    if (String(topic) == TOPICO_COMANDO_RACAO && msg == "ON") {
        digitalWrite(LED_RACAO, HIGH);
        delay(3000);
        digitalWrite(LED_RACAO, LOW);
    }
    if (String(topic) == TOPICO_COMANDO_AGUA && msg == "ON") {
        digitalWrite(LED_AGUA, HIGH);
        delay(3000);
        digitalWrite(LED_AGUA, LOW);
    }
    if (String(topic) == TOPICO_COMANDO_TEMPERATURA && msg == "ON") {
        digitalWrite(LED_TEMPERATURA_AGUA, HIGH);
        delay(3000);
        digitalWrite(LED_TEMPERATURA_AGUA, LOW);
    }
```

```
}

void verificaSensorRacao(unsigned long tempoAtual) {
    if (digitalRead(BOTAO_RACAO) == LOW && tempoRacao == 0) {
        Serial.println("Simulando sensor: Falta de racao");
        MQTT.publish(TOPICO_ALERTA_RACAO, "FALTA");
        tempoRacao = tempoAtual;
    }
    if ((tempoAtual - tempoRacao) >= intervalo && tempoRacao > 0) {
        MQTT.publish(TOPICO_ALERTA_RACAO, "NORMAL");
        tempoRacao = 0;
    }
}

void verificaSensorAgua(unsigned long tempoAtual) {
    if (digitalRead(BOTAO_AGUA) == LOW && tempoAgua == 0) {
        Serial.println("Simulando sensor: Falta de agua");
        MQTT.publish(TOPICO_ALERTA_AGUA, "FALTA");
        tempoAgua = tempoAtual;
    }
    if ((tempoAtual - tempoAgua) >= intervalo && tempoAgua > 0) {
        MQTT.publish(TOPICO_ALERTA_AGUA, "NORMAL");
        tempoAgua = 0;
    }
}

void verificaSensorTemperatura(unsigned long tempoAtual) {
    if (digitalRead(BOTAO_TEMPERATURA_AGUA) == LOW && tempoTemperatura == 0)
    {
        Serial.println("Simulando sensor: Temperatura da agua alta");
        MQTT.publish(TOPICO_ALERTA_TEMPERATURA, "ALTA");
        tempoTemperatura = tempoAtual;
    }
    if ((tempoAtual - tempoTemperatura) >= intervalo && tempoTemperatura >
   0) {
        MQTT.publish(TOPICO_ALERTA_TEMPERATURA, "NORMAL");
        tempoTemperatura = 0;
    }
}
```

Listing 1: Código do Trabalho 5 - Simulação de Sensores e Atuadores

A.2 Anexo B: Código Fonte do Backend (Python)

```
# Codigo do Trabalho 6 - Persistencia em Banco de Dados
import paho.mqtt.client as mqtt
import mysql.connector
import time

# Configuracoes MQTT
BROKER = "test.mosquitto.org"
PORTA = 1883
TOPICOS = ["/pet/comando/temperatura", "/pet/alerta/temperatura", "/pet/
   alerta/racao", "/pet/comando/racao", "/pet/alerta/agua", "/pet/comando/
   agua"]
ID_CLIENTE = f"PythonMQTT_{int(time.time())}"

# Conexao com MySQL (usando Docker/AWS)
conexao = mysql.connector.connect(
    host="localhost", # ou o endpoint da AWS
```

```
    user="user_admin",
    password="passwd_admin",
    database="trabalho_db"
)
cursor = conexao.cursor()

# Funcoes de Callback
def on_connect(client, userdata, flags, rc):
    if rc == 0:
        print("[MQTT] Conectado com sucesso!")
        for topico in TOPICOS:
            client.subscribe(topico)
            print(f"[MQTT] Inscrito no topico: {topico}")
    else:
        print(f"[MQTT] Falha na conexao. Codigo de erro: {rc}")

def on_message(client, userdata, msg):
    mensagem = msg.payload.decode()
    topico = msg.topic
    print(f"[MQTT] Mensagem recebida: {mensagem} | Topico: {topico}")

    # Comandos especiais para controle do script
    if mensagem.lower() == "termina":
        print("[MQTT] Comando 'termina' recebido. Encerrando...")
        client.disconnect()
        conexao.close()
        exit()
    elif mensagem.lower() == "delete":
        print("[MQTT] Comando 'delete' recebido. Limpando tabela...")
        cursor.execute("TRUNCATE TABLE dadosIoT")
        conexao.commit()
        return

    # Insere a mensagem no banco de dados
    cursor.execute(
        "INSERT INTO dadosIoT (topico, mensagem) VALUES (%s, %s)",
        (topico, mensagem)
    )
    conexao.commit()
    print("[DB] Dados inseridos com sucesso.")

# Execucao do cliente MQTT
cliente = mqtt.Client(client_id=ID_CLIENTE)
cliente.on_connect = on_connect
cliente.on_message = on_message

cliente.connect(BROKER, PORTA, 60)
print("[Sistema] Aguardando mensagens... (Ctrl+C para sair)")
cliente.loop_forever()
```

Listing 2: Código do Trabalho 6 - Persistência em Banco de Dados

A.3 Anexo C: Interface do Aplicativo Mobile

A imagem abaixo mostra a tela principal do aplicativo IoT MQTT Panel, configurada para interagir com o sistema. É possível visualizar os alertas de ração, água e temperatura, além de enviar comandos para liberar ração e água manually.

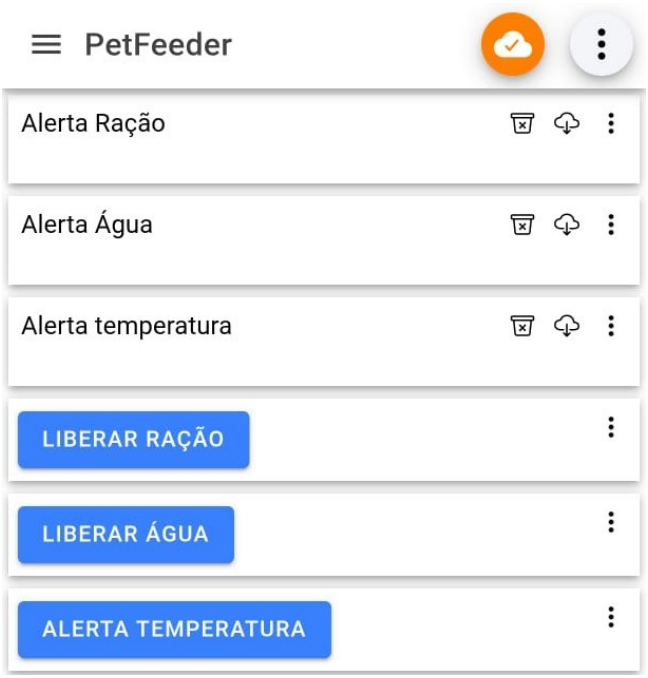

Figura 3: Interface do aplicativo IoT MQTT Panel configurada para o projeto.